Hartley Hyde
cactus.pages@internode.on.net

Software priorities

The early eighties saw a period of rapid change in computing and teachers lost control of how they used computers in their classrooms. Software companies produced computer tools that looked so good that we forgot about writing our own classroom materials and happily purchased software — that offered much more than we needed — from companies that had little knowledge of pedagogy. This article encourages you to budget time and money for a better balance between teaching how to use powerful computer tools and writing your own materials to meet the needs of your classes.

In the sixties, there was only one item on our school computer education budget: computer cards. The Computing Centre supplied all of the other computing materials, including huge quantities of tractor feed paper. The system software was supplied with the computer. The educational software was written and shared by teachers.

In the late seventies, we budgeted to buy microcomputers, floppy disks, printers and paper but there was still no thought of buying software. The writing of software was a leisure time activity and was seldom seen as "part of the job." We were enthusiasts and we shared what we wrote. The software was therefore free, easily copied and easily modified to meet the needs of different classes.

So how did we use computers? Taylor (1980) identified three modes of computer education where the computer functions as a tool, as a tutor, or as a tutee or student.

The most complicated tool I wrote helped me build school timetables. That can now be done using dedicated timetabling applications, database managers or spreadsheets, but these tools were not then available.

A computer was used in tutee mode when students wrote small problem solving programs using *BASIC* or *APL*. Later we preferred *Pascal*, *Logo* and eventually *JavaScript* because these languages encouraged good structure. We reasoned that if students understand a task (such as solving a quadratic equation) well enough to program a computer to consistently obtain the correct answer, then they have spent so much time working through all the possibilities that they have covered the topic thoroughly.

This type of computer use is now rare because programming using BASIC or Pascal got much harder, few teachers know how to program and because programming tends to hog machines which can be used for more important tasks such as essay writing, surfing the Net, sending e-mails or downloading games and music. However, if you encourage your students to program using JavaScript, they can do most of their work at home and their enthusiasm to build clever web pages provides strong motivation.

Another form of tutee activity centred on a use of micro-worlds such as the turtle-graphics part of Logo. The more complex features of Logo were seldom taught. We did not need to buy Logo because there were several free look-alikes that offered only turtle-graphics.

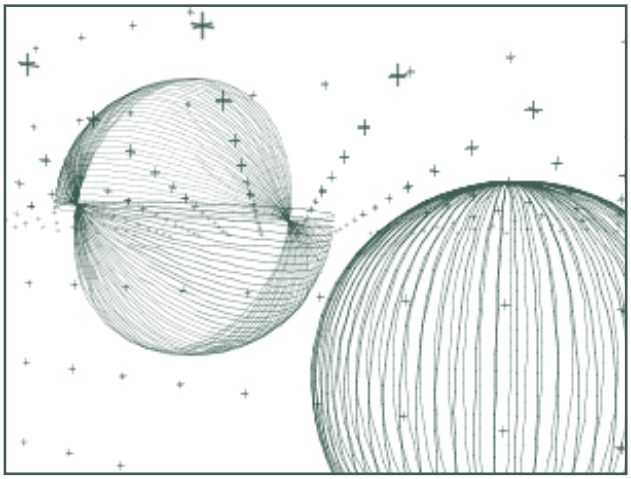

Figure 1. 3D Logo.

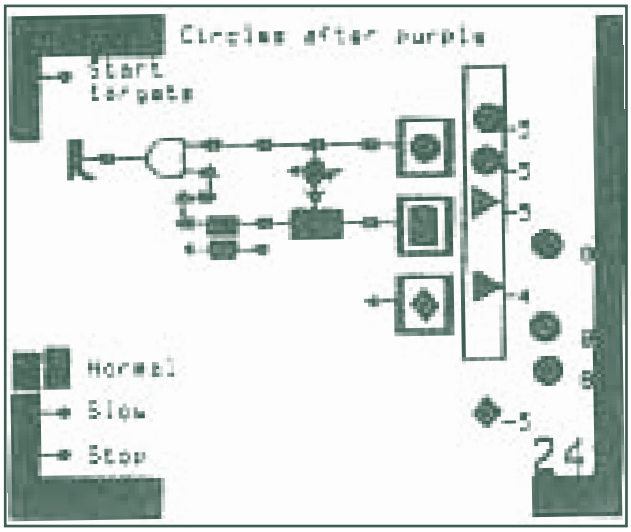

Figure 2. Rocky's Boots.

Another popular micro-world was *Rocky's Boots* in which students had to build a virtual machine using logic gates — but by that time we had to pay for good software.

Taylor would classify the rest of our computer usage as tutor mode. There were many types of tuition software. Much of the better tuition software was presented as a game and the tests were called quizzes. Microcomputers allowed demonstrations of animated models. The Angle Park Computing Centre, Adelaide, specialised in simulations of scientific or mathematical topics, which could be built on a mathematical model.

I particularly liked drill and practice software because I found drilling students quite boring. Why perform dull repetitive tasks yourself when a computer can do it better? A computer can be programmed to concentrate on material that the individual student finds difficult, it gives immediate feedback and it is perceived as non-judgmental.

Tutorial software is not difficult to design, it can be modified to suit the needs of each class and by using authoring software like *PILOT* most teachers can learn to write their own. The latest version of *PILOT* can produce smart interactive web-pages that can be used from home or school.

But the winds of change had been blowing. In 1976 a young man named Bill Gates had written an open letter to the Homebrew Computer Club accusing members of theft because they had shared copies of Altair 6800 *BASIC*. You can view a copy of the letter on the Wikipedia. Despite the angry replies, Gates persisted. Good software would only be written if people were prepared to pay for it.

The first "killer application" was *Visicalc*. 1979–1983 saw the worst drought in Victorian history. The Department of Agriculture took a bold step. They wrote simple *Visicalc* spreadsheets that allowed farmers to calculate the optimum number of sheep or cattle to cull. If they killed too few, the food would run out and all the animals would perish. Teachers from country centres were taught to use the package and they took a computer, *Visicalc* and the spreadsheets back to their school. At night they helped farmers calculate how many animals to kill. This was the most compelling example of using a computer tool that I had seen.

The worldwide impact of *Visicalc* was enormous. For the first time we saw software that looked useful and worth buying. However, *Visicalc* was fairly easily copied and could be acquired too easily in a culture which still thought software should be free.

Other expensive tools were soon released. In 1980, several database management systems appeared. By 1983 a variety of word processors could be purchased for each type of machine and software for remote terminal access became available.

When he reviewed the state of computer education in Australian schools, Anderson (1984) described situations where all of Taylor's modes were being used in our schools; but the early eighties saw rapid change in so many aspects of computer use that natural selection was bound to occur.

Change was accelerated when eight-bit machines such as Apple IIs and BBCs were replaced with sixteen-bit Macintosh and Windows machines. Although there had been thousands of tutorial packages for eight-bit machines, very few of these were re-coded for the new machines because the task was too challenging. Some of this type of software has since been coded using *Java* or *JavaScript* and can be accessed using the Internet.

Some new tutorial software has come on the market and most of the examples I have seen show that the authors have not done their homework. Most of what we knew about the psychology of computer education and how to write effective tutorials seems to have been lost or ignored. There is not much money to be gained by selling tutorials to schools, so many of the new tutorials are aimed at the concerned parent market.

Programming the new platforms using *Pascal* or *BASIC* was a much more complex task. The only tutee environments easily available were *Logo* and *HyperCard* scripts and even using graphics environments the results were disappointing compared with the games students were buying for their PCs. While some of us continued to teach programming, a tutee mode of instruction did not blossom in the same way that teachers took to using computers as tools.

When software suppliers offered attractive deals, schools around the world surrendered a significant proportion of their instruction time to training students how to use those tools. This meets with parent and government approval because it is perceived as preparing students for the "real world." Obviously we have to teach how to use the tool first, but the medium should not be the message. Unless a use of computers eventually leads students to learn aspects of our curriculum in new and more powerful ways, we are wasting valuable instruction time.

In contrast, the various geometry packages have been designed for student use, the menu system is simple and logical and little time is needed to teach how to use the tool. When we use these packages we expect our students to learn geometry.

When I first saw a graphics calculator, I thought that it was a step backward into the Apple II era. A new generation of graphics calculators now comes with a useful set of mathematical tools that have been designed for students. When we use them, we do not expect to spend very much time teaching how to use the tools. We go into the classroom expecting to teach mathematics.

Some of the new graphics calculators also come with a potential to develop our own software. Teachers who use *ClassPads* will know that they can program many of the features using *BASIC*. This offers a potential to re-introduce a tutee mode of teaching. Once hooked, some students will spend much time enjoying their newfound mathematical power.

There may even be some of you who discover that you can develop simple calculator software that actually teaches students mathematics — it is possible to write tuition software for a *ClassPad*. You could share with other teachers world wide using the Internet. The software would be free and it could be modified to suit different classes.

You may apply the same thinking to your use of computers. In previous articles I have shown how to program using *JavaScript* and how to use *JavaSketchPad* from within the browsers that come with your system software. By including a use of *PILOT* it is fairly easy to produce smart web pages that can tutor students at home or at school. All you need is time.

When you gain confidence producing your own materials, it is not that much harder to take secondary students with you. We know that students learn from drill and practice programs most effectively when they believe they have written the programs themselves.

If I were preparing the computing section of a mathematics budget, my highest priority would be buying conference time so that my staff could learn how to use the software they already have and to use it more effectively.

I believe we lost control of our use of computers in the early eighties when we allowed teaching *how to use computer tools* become more important than tutor and tutee modes of instruction. A new generation of teachers is capable of regaining control by learning how to write their own computer materials and how to share this skill with their students. Therefore my second priority would be finding time and ways to resource and encourage volunteers to rediscover the potential and empowerment of learning how to program using their graphics calculator, *PILOT* or *JavaScript*. This is just a first step toward revitalising the traditional and then perhaps discovering new modes of computer education.

References

Anderson, J. (1984). *Computing in schools: An Australian perspective*. Hawthorn: ACER.

Taylor, R. P. (Ed.) (1980). *The computer in the school: Tutor, tool, tutee*. New York: Teachers College Press.